1. Complete as lacunas transformando a classe Albergue em um *Singleton*.

```
public class Albergue {

    private String nome;

    private static final Albergue albergue = new Albergue();

    ____private____ Albergue() {

    }

    public ____static Albergue____ getAlbergue() {

        return ____albergue____;

    }
}
```

0,5
2,5
1,0
3,0
1,5
0,0
0,0
0,5
9,0

2. Sendo fornecida a classe `Albergue` abaixo, preencha as lacunas de acordo com as perguntas 2.1 a 2.5 (enunciado após o código):

```
public class Albergue {
    private String nome;

    private Reserva[] reservas = new Reserva[3]; // 2.1

    private static int contador; // 2.2

    public boolean adicionarReserva(String dataReserva) {

        Reserva reserva = new Reserva(); // 2.3

        ; // 2.4
```

}

}

9,0
0,5
9,5

2. Sendo fornecida a classe `Albergue` abaixo, preencha as lacunas de acordo com as perguntas 2.1 a 2.5 (enunciado após o código):

```
public class Albergue {
    private String nome;
    private Reserva[] reservas = new Reserva[3]; // 2.1
    private static int contador; // 2.2

    public boolean adicionarReserva(String dataReserva) {
        Reserva reserva = new Reserva(); // 2.3
        reserva.setDataReserva(dataReserva); // 2.4
        reservas[contador] = reserva; // 2.5
        contador++; //2.5

        return true;
    }
}
```

2.1 Um albergue possui 0, 1 ou mais objetos do tipo `Reserva` atribuídos a uma variável de referência chamada `reservas`, representadas por meio de um *array* de tamanho 3.

2.2 Um contador é necessário para sabermos o ponto de inserção de um objeto do tipo `Reserva` dentro do *array* da Questão 2.1.

2.3 Deve-se instanciar um objeto `Reserva` (suponha que exista uma classe chamada Reserva com um construtor sem argumentos) e atribuí-lo a uma variável de referência chamada `reserva` (esta variável é usada na próxima questão).

2.4. O objeto criado na questão anterior deve ser configurado com os valores recebidos no método `adicionarReserva( )` (obs: suponha que `Reserva` possua uma variável de instância do tipo `String` chamada `dataReserva` com seus respectivos métodos `getDataReserva( )` e `setDataReserva(String dataReserva)`.

2.5 Qual(is) instrução(ões) deverá(ão) ser inserida(s) para adicionar uma reserva no *array* da Questão 2.1, mantendo o correto funcionamento do controle de reservas?

3. Sendo fornecida a classe `Hospede` abaixo, preencha as lacunas de acordo com as perguntas 3.1 e 3.2 (enunciado após o código):

```
public class Hospede {
     private String nome;
     private String sobrenome;

     public Hospede()
        this("");                                          ; // 3.1
     }

     public Hospede(String nome) {
        this(nome, "");                                    ; // 3.1
     }

     public Hospede(String nome, String sobrenome) {
        this.nome = nome;                                  ; // 3.2
        this.sobrenome = sobrenome;                        ; // 3.2
     }
}
```

3.1 Repare que na classe `Hospede`, existe sobrecarga de construtores. Qual código deve ser inserido nas DUAS lacunas associadas a esta questão de forma a se iniciar recursivamente um objeto `Hospede`?

3.2 As variáveis de instância devem ser iniciadas nestas duas lacunas resolvendo possíveis ambiguidades entre nomes de variáveis de instância e locais ao construtor.

4. Sendo fornecida a classe `Pagamento` abaixo, preencha as lacunas de acordo com as perguntas 4.1 e 4.7 (enunciado após o código):

```
public ____abstract class____ Pagamento {  // 4.1

    ____protected____ double valorHospedagem; // 4.2

    public Pagamento CriarPagamento(String tipo) {
        Pagamento pagamento = ____null;____ // 4.3
        switch (tipo) {
            case "Dinheiro" :

                ____pagamento = new PagamentoDinheiro();____ // 4.4
                break;

            case "Cartao" :

                ____pagamento = new PagamentoCartao();____ // 4.4
                break;

            case "Cheque" :

                ____pagamento = new PagamentoCheque();____ // 4.4
                break;
        }

        return ____pagamento____ // 4.5
    }

    @Override
    public String toString() {
        ____System.out.println("Valor da hospedagem: R$" + ____ // 4.6
        ____valorHospedagem);____ // 4.6
        ____                ____ // 4.6
    }

    public void setValorHospedagem(____double valorHospedagem____) { //4.7
        ____this.valorHospedagem = valorHospedagem;____ // 4.7
    }
}
```

4.1 `Pagamento` é uma classe que não pode ser instanciada.

4.2 Modificador de acesso associado à herança, que permite que subclasses de `Pagamento` possam acessar diretamente a variável de instância `valor`.

4.3 Variáveis locais não são iniciadas automaticamente. Qual código pode ser colocado aqui para evitar erro de compilação? (observação: a resposta não é código para criação de um novo pagamento – Isso vai ser discutido na próxima questão).

4.4 Existem 3 subclasses de `Pagamento` (`PagamentoDinheiro`, `PagamentoCheque`, `PagamentoCartao`) Todas as subclasses possuem um construtor default (que não recebe argumentos). Elas são criadas neste método com base no argumento do método (`tipo`) e na variável local `pagamento`.

4.5 Que código deve ser inserido aqui?

4.6 Qual código deverá ser inserido aqui para que seja possível ser exibido no console o valor de um pagamento precedido da sigla R$ (não necessariamente a solução tem 3 linhas como no trecho de código e sinta-se à vontade em usar o operador + na sua resposta, em vez de usar uma classe para formatação de valores monetários, como visto em Laboratórios).

4.7 Quais códigos devem ser inseridos nestas duas lacunas?

5. Sendo fornecida a classe `PagamentoDinheiro` abaixo, preencha as *lacunas de acordo* com as perguntas 5.1 a 5.3 (enunciado após o código):

```
public class PagamentoDinheiro extends Pagamento { //5.1

    private double valorFornecido;

    public PagamentoDinheiro(double valorHospedagem, double valorFornecido){
        super(valorHospedagem); //5.2
        this.valorFornecido = valorFornecido; //5.3
    }
}
```

5.1. Pagamento em dinheiro é um tipo possível de pagamento.

5.2 Qual código deverá ser inserido aqui para iniciação da variável de instância `valorHospedagem` presente na classe `Pagamento`? (suponha que `valorHospedagem` possua modificador de acesso `private` e que exista um construtor corretamente implementado na classe `Pagamento` que recebe `valorHospedagem` como argumento).

5.3 Qual código deverá ser inserido aqui?

6. Na orientação a objetos uma classe abstrata é construída para ser um modelo para classes derivadas e na sua construção há algumas restrições. Assim, considere uma classe abstrata de nome `Calculo`. A instrução que NÃO é permitida nessa classe:

a. `private abstract int multiplicar(double n1, int n2);`
b. `public abstract void exibirResultado();`
c. `public Calculo() {}`
d. `private static final double VALOR=10;`
e. `private double multiplicar(double n1, double n2){return n1*n2;}`

7. Qual o resultado ao se executar este programa? (as reticências indicam código omitido. O discente deverá assumir que este código omitido foi codificado corretamente).

```
class AtribuicaoReferencia {
   private String nome;

   AtribuicaoReferencia(String nome) {
      this.nome = nome;
   }

    public void setNome(String nome) {
       this.nome = nome
    }

     public String getNome() {
        return nome;
     }

     public static void manipularReferencia(AtribuicaoReferencia r1, AtribuicaoReferencia r2) {
        r1.setNome("Gregório");
        r2 = r1;
        r2.setNome("Ana");
     }

     public static void main(String[] args) {
        AtribuicaoReferencia p1 = new AtribuicaoReferencia("Professor");
        AtribuicaoReferencia p2 = new AtribuicaoReferencia("Estudante ");
        manipularReferencia (p1, p2);
        System.out.println(p1.getNome() + "," + p2.getNome());
     }
}
```

a. Exibe: Professor, Professor
b. Exibe: Estudante, Professor
c. Exibe: Gregório, Estudante
d. Exibe: Ana, Estudante
e. Nenhuma das demais alternativas.

8. O código Java abaixo não compila. Por quê? como corrigir?

```
public class Teste {

   public static void main(String[] args) {
     metodo1();
   }

   public void metodo1(){
     System.out.println("não compila");
   }
}
```

O código não funciona porque o método "metodo1" não é "static", ou seja, é necessário criar um objeto da classe "Teste" para usá-lo. Assim, para corrigi-lo, basta transformar o método "metodo1" em "static", ficando assim:

CORRIGIDO

```
public class Teste {
    public static void main(String[] args){
        metodo1();
    }

    public static void metodo1(){
        System.out.println("não compila");
    }
}
```

Boa Prova !
Rodrigo Martins Pagliares